Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

User's Manual

Hybrid Battery Pack for Self-Launching eGliders

Submitted to

CyPi Ltd.
Dr. Kenneth D. Sebesta
859-648-0272
kenn@cypiltd.com

by

Team #23
HYP Batteries

Team Members

Spencer P. Piligian, spiligia@bu.edu
Daniel Kao, dkao1234@bu.edu
Yusheng Chen, chenyush@bu.edu
Steven Wang, swang3@bu.edu
Ismaeel AlAlawi, ialalawi@bu.edu

Submitted: April 18, 2022

Hybrid Battery Pack for Self-Launching eGliders

Table of Contents

Executive Statement

Batteries for eGliders are a niche market, because they demand performance similar to that required for a rocket launch while still needing to be lightweight enough for a small aircraft. As a result, there are currently no cheap commercial off-the-shelf battery packs suitable for an eGlider. Our hybrid-chemistry battery pack design aims to fill this gap by combining two batteries with two different discharge characteristics. These two cell chemistries can achieve a variable-max discharge rate, allowing us to design a smaller battery pack while also fulfilling power requirements during the launch. This pack setup will help eGliders achieve a safe flight through the use of electric power and as a result, will minimize the risk of mechanical failure for our client's conventional mechanical engines, whose vibration causes other parts of the aircraft to wear out and ultimately increases the chance of failure.

1 Introduction

An eGlider's self-launch profile is much like that of a rocket's: both require a period of high thrust followed by discrete reductions in power output.

1. Peak-power ground acceleration (0m/s -> 25m/s)
2. Peak-power climb to Safe Altitude (0m -> 100m)
3. Max continuous-power climb to Airport Pattern Altitude (100m- > 300m)
4. Gentle climb to Thermal Altitude (300m -> 500m)
5. Level flight at Thermal Altitude (500m)

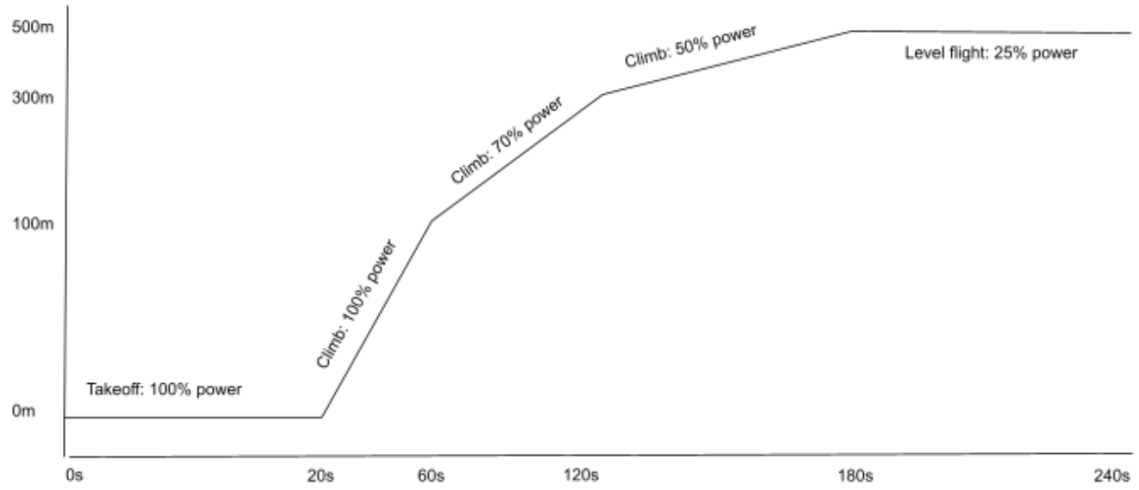


Figure 1: Self-launch profile of an eGlider.

This self-launch lasts approximately four minutes. Based on the launched profile prediction we made, we can estimate the power use at different stage launching. Ideally, the battery pack will use all its energy by the end of the launch. Reaching the desired altitude with more energy capacity than is necessary *maximizes* weight, cost, and fire risk – all of which are undesirable traits in aviation.

Current battery technology does not allow for a small, safe battery pack capable of meeting the rocket-launch requirement: lithium iron phosphate (LFP) batteries are too fire prone, and individual lithium-ion (Li-ion) batteries do not have the ability to source the intense currents required for launch on their own.

The current solution is to install a battery pack which is much larger than ideal. This larger pack allows Li-ion cells to share the current load, bringing the per-cell current

to a survivable 2-4C discharge rate. However, the downside is a pack which only uses 20% of its capacity for launching, which means the pack is 70-80% heavier than optimal.

A hybrid LFP / Li-ion pouch cell pack delivers a blend of power and energy. The LFP is capable of sourcing more than 100A for short bursts, though it does not have great energy density. The Li-ion pouch cell has great energy density, but cannot source more than 30A per cell.

In a cell series count ratio of 3:2, simulations show that the desired power properties are evident, and the proposed pack weighs two to three times less than the current state-of-the-art.

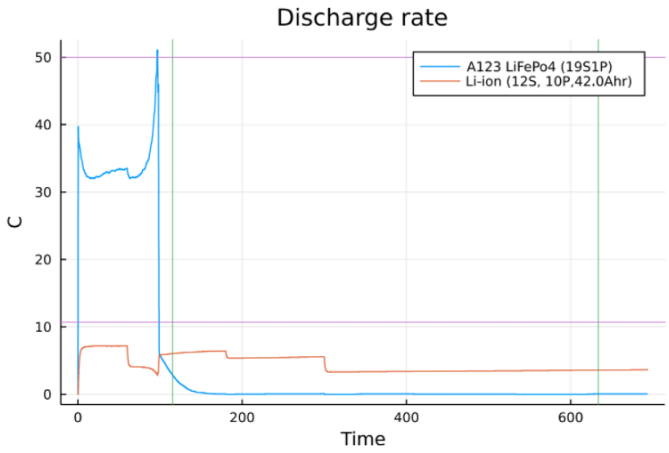


Figure 2: Julia software simulation.

Please carefully read and follow the instructions to ensure the safe use of our battery pack. Since the battery pack can go over 60V and capable of discharging at 300A, failure to follow the instruction may cause injury and cause damage to the battery pack.

The following sections will provide deeper descriptions on the system and give instructions on how to operate the battery pack appropriately.

2 System Overview and Installation

2.1 (a) Overview block diagram

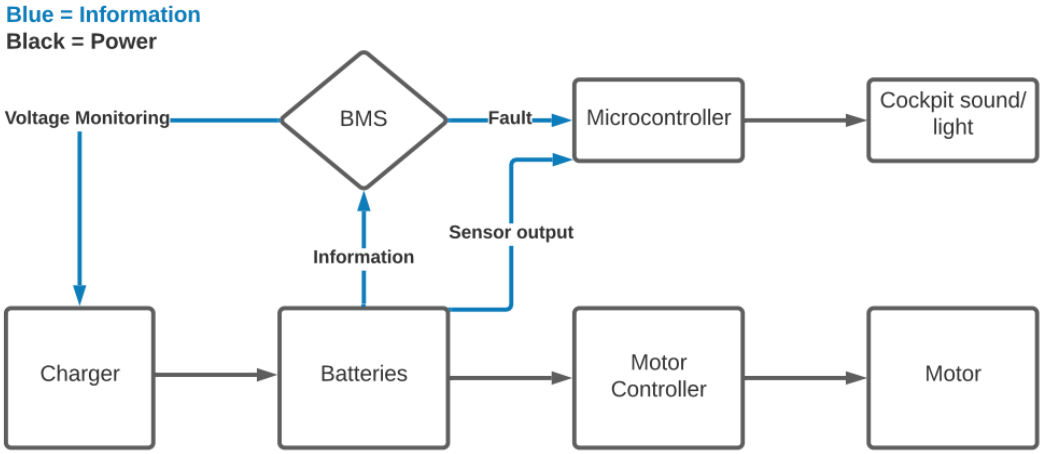


Figure 3: System overview block diagram with labeled functional and information flow lines.

As per the figure above, the block functions of the battery pack are split into two main categories: information and power. While the batteries are being charged, the voltage across the pack will be monitored to ensure they are not overcharged. During flight, the pack will also have a BMS and temperature monitoring system to manage the temperature of cells in the pack. This information will be passed to a microcontroller that the pilot can read externally. The second path of the block diagram is the cell power. The batteries will power a motor controller that manages the power input to the motor.

2.1 (b) Overview Automated Testbench Software Architecture

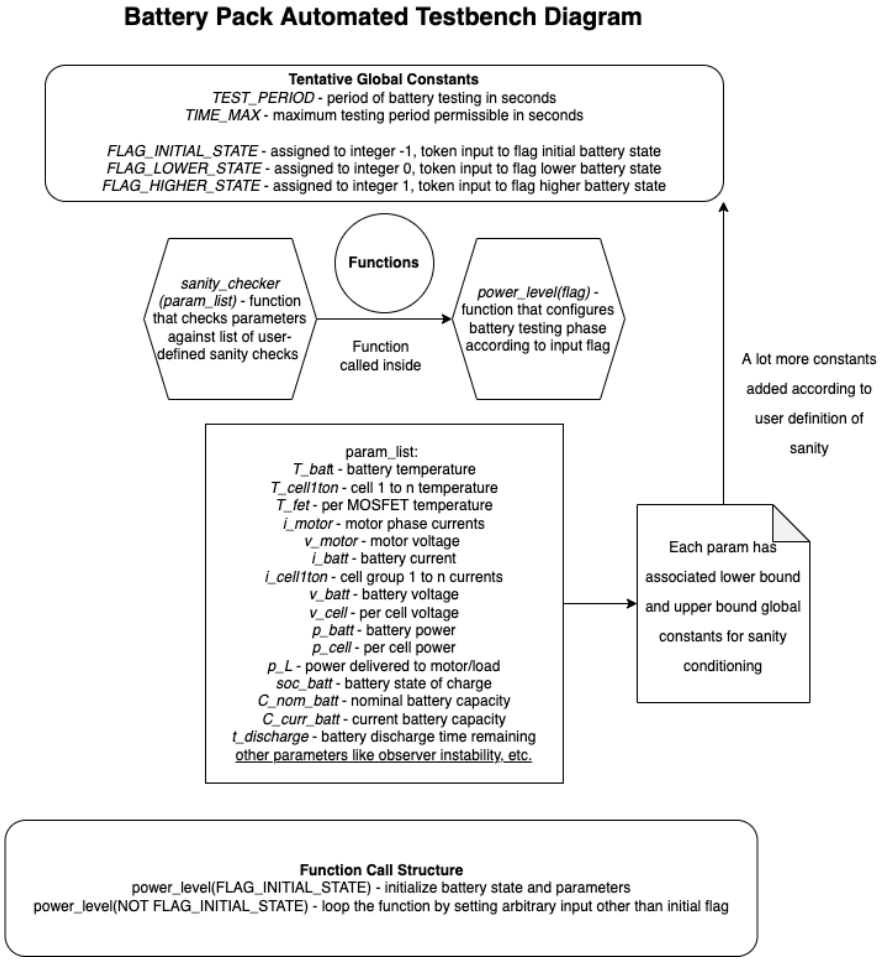


Figure 4: Battery pack automated testbench diagram using modified UML (Universal Modeling Language) syntax.

The testbench code should have some loop with at least four "top-view" conditions (a.k.a. battery monitoring sanity checks) and then it should have some actuation mechanism where as long as those conditions are satisfied, then the next phase of testing will begin according to the state variable or parameter list. So, the top view diagram provided above lists the definitions and relations between the following relevant **battery testing parameters**:

- *t_test* - Testing period (time set for testing that is measured in seconds and ranges from 0 to TIME_MAX), where TIME_MAX is some constant that is defined by the user or client accordingly
- *user_stop* - User-defined input (manual STOP command of testing)
- *sane & insane* - Sane state is that which meets the conditions involving each parameter outlined below, and insane is that which doesn't otherwise

- *T_batt* (battery pack temperature, however it is defined), *T_cell1, T_cell2,..., T_celln* (temperature of each cell as defined by the engineers)
- *T_fet* (MCU MOSFET temperature)
- *i_motor* (motor phase currents)
- *v_motor* (motor voltage)
- *i_batt* (battery current), *i_cell1, ... , i_celln* (given cell group current)
- *v_batt* (battery voltage) and *v_cell* (per cell voltage)
- *p_batt* (battery power), *p_cell* (per cell power), *p_L* (current power delivered to load/motor)
- *SOC_batt* (battery state of charge)
- *C_nom_batt* (nominal capacity of battery), *C_current* (current capacity of battery)
- *t_discharge* (time remaining for battery to discharge)

As for the **functions** used in the diagram:

- *power_level(flag)* - A power level function that sets relevant parameters according to some input token (“flag”)
  - the power level function determines the initial state of the battery and sets the relevant parameters (initial values)
  - the input token determines whether or not the test transitions to a higher or lower power level or shuts down
  - the function prints relevant parameters to the console for debugging and even manipulate some data if needed
- *sanity_checker(param_list)* - A sanity check function that evaluates the current battery state and set its values according to a number of conditions
  - the sanity checks are conditions that are defined such that some flag token is set to some value corresponding to sane or insane
  - either power level function is called in the sanity check function (or vice-versa possible) to parse the sane/insane flag inputs and transition the power level according to that flag token. If the flag input is sane, then power is transitioned to higher level, and otherwise, if it's insane, then the power is transitioned to lower level
  - in the definition of the sanity check function, the function itself is called again in order for it to loop indefinitely according to some sleep frequency (similar to void loop in Arduino IDE)

After defining the power level function and sanity check function in the lisp script, the power level function would be first called with the corresponding input token for initializing the battery state and determining its relevant parameters (i.e. it runs once similar to void setup in Arduino IDE). Then, the sanity check function would be directly called with the outputs of the power level function as its inputs.

2.2 User interface

Our user interface will be a simple LED indicator which will signal any errors in the pack. When the battery management system or temperature sensor detects any problems in the pack, the fault will be sent to the microcontroller which will prompt the LED indicator to turn on. This indicator will be placed in the cockpit so the pilot will be aware of any issues during flight.

A separate, second indicator will be used for charging the batteries. This indicator will turn green if the battery is charging properly. If the battery cells have a voltage imbalance, the indicator will change color notifying the user of an error. This signal will be communicated from the microcontroller managing the charging and sent to the LED indicator.

2.3 Physical description

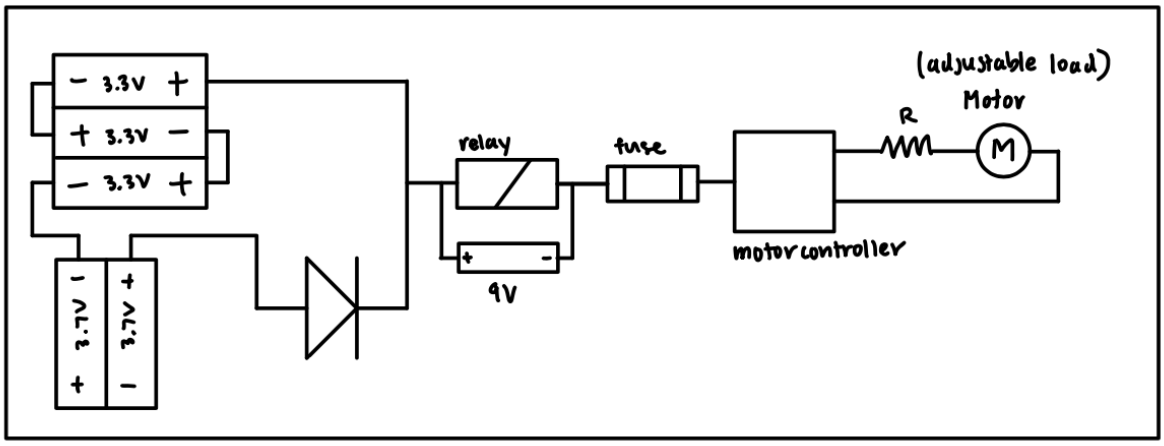


Figure 5: Final battery testing schematic.

2.4 Installation, setup, and support

The following **software packages** are needed before proceeding with the software configuration instructions:

- VESC Tool (free version) GUI developed by Benjamin Vedder
  - Qt Toolkit
  - .zip file extractor (Windows, Linux, Mac OS X, and Android supported like WinZip or 7-Zip)
  - Source code: https://github.com/vedderb/vesc_tool
- VESC-Lisp, Common Lisp, or LispBM Scripts

    - ANSI Common Lisp GNU v2.49 (2010-07-07 build) compiler (REPL)
    - For Mac OS X, install via MacPorts
    - For Windows and Linux (Ubuntu, Debian, Sun Solaris, etc.), install directly from https://clisp.sourceforge.io/

- Arduino IDE v1.8.19
    - Supported for Windows 7+, Linux (32-bit & 64-bit), Mac OS X (10.10 or newer), ChromeOS
    - Download directly from https://www.arduino.cc/en/software
- Arduino C/C++ Scripts (sketches)
    - Firmata library for all boards
    - *LiquidCrystal.h* library – optional for LCD display of temperature logs

The following **pre-testing instructions** must be followed to assemble the hardware and configure the software:

- *Battery testing circuit hardware side*
    - Connect 12 V lab DC power supply to the bottom left terminal of the 30 A relay using alligator or crocodile clips
    - Connect 9 V battery to the relay (top left and right connections) by soldering battery snap connectors to power it up and connect other 9 V battery to the current sensor using battery snap connectors to also power it up
    - Place each of the three LFP batteries in their respective single slot battery holders
    - Place both Li-ion batteries in the dual slot battery holder
    - Solder the wires connecting the LFP battery holder contacts in series and solder the first ring terminal wire to one of the LFP battery holders accordingly and the other to the Li-ion dual battery holder to create the parallel connection
    - Connect the ring terminal connected to the LFP battery holders into terminal ~C (phase-C) of the rectifier and the other terminal (which is connected to Li-ion batteries) to terminal B-
    - Solder the LFP battery holder and rectifier junction into the 30 A relay's bottom right terminal
    - Solder the 7 A fuse into the bottom-right terminal of the Li-ion battery holder and connect its other end to the negative top terminal of the current sensor
    - Connect the positive top terminal of the current sensor to the motor controller positive XT30 connection; the negative terminal of this

connection should be connected to the multimeter accordingly for voltage readings
    - Connect the positive terminal of the motor controller MR30 connector into the positive terminal of the multimeter to complete the connection for motor controller voltage readings
    - Connect the A50S motor controller to the laptop using micoUSB-USB cable
    - Connect the pico-clasp connector shell to MR30 connector, and connect that to the brushless motor's connector
    - Connect the positive and negative terminals of the motor controller-motor junction to the positive and negative terminals of the power resistor respectively
- *Adafruit Huzzah ESP32 microcontroller*
    - Connect ESP32 to computer with microUSB-USB cable (if using a laptop with only USB-C ports, then a USB to USB-C hyperdrive or an adapter)
- *Arduino Nano microcontroller hardware side*
    - Connect Arduino Nano to computer with microUSB-USB cable (if using a laptop with only USB-C ports, then a USB to USB-C hyperdrive is needed)
    - Since resistors are blind to polarity, the thermistor and resistor ends are arbitrary, so for clarification, let one end be (a) and the other be (b) for both respectively
    - Connect pin #27 (5V DC power) to end (a) of 10K resistor, and connect end (b) of the resistor to end (a) of thermistor using male-to-male jumper wires
    - Connect pin #19 (A0/D14) to end (a) of thermistor using male-to-male jumper wires
    - Connect pin #29 (GND) to end (b) of thermistor using male-to-male jumper wires
- *VESC Tool setup software side*
    - Create an account on https://vesc-project.com/ > click on "VESC tool' tab > purchase VESC Tool Free for €0.00 (no billing information required) > click on "Purchased Files" link (top right corner) > download and extract "vesc_tool_BETA_ALL.zip" (or for Windows users, download "vesc_tool_free_windows.zip")
    - Alternatively install directly via source code: https://github.com/vedderb/vesc_tool
    - For Windows and Linux users, Install REPL Lisp compiler via SourceForge
    - For Mac OS X users install MacPorts v2.7.1

        - Install Xcode and Xcode Command Line Tools
        - Agree to Xcode license in Terminal: sudo xcodebuild -license
        - Install MacPorts according to version of OS X (convenient installation through .pkg installer for version 10.15+) on https://www.macports.org/install.php
        - After MacPorts installation, type sudo port install clisp to install common lisp implementation (interpreter, compiler, debugger, etc.)
- *Arduino IDE software side*
    - Go to Tools > Manage Libraries > Drop-down "Type" > Installed. Double-check that Firmata built-in library is uncorrupted and installed properly. For LCD temperature display option, if not already installed, search for LiquidCrystal and install version 1.0.6
    - For microcontroller testing, go to Tools > select Boards Manager > search for Arduino AVR Boards > ensure most up-to-date version installed (v1.8.5 as of in-lab testing) > select Board: Arduino Nano > under same tab, select appropriate serial port when board is connected
    - Once Arduino board is properly connected with aforementioned hardware configuration on breadboard, run the temperature logging Arduino code and check the serial monitor for logged temperature readings – modify sampling rate as appropriate. As a quick test, check that room temperature is properly recorded
- *Qt Creator*
    - Install Qt Creator using an open source development link. The link will automatically detect between Mac OS or Windows user: https://www.qt.io/download-qt-installer?hsCtaTracking=99d9dd4f-5681-48d2-b096-470725510d34%7C074ddad0-fdef-4e53-8aa8-5e8a876d6ab4
    - Open *BatteryDisplay* file after successful installation
    - Build and run the app
    - Make sure you compile instruction for data you want to send through BLE on the EPS32 using Arduino IDE
    - Scan for device on the *BatteryDisplay* app and connect to ESP32 through BLE
    - Check to see if data are being received from ESP32, make sure the command execution are sent to the ESP32 through Arduino
- *Diode, 9 V batteries, and properly soldered electrical connection testing (if necessary)*
    - Test the voltages/currents (using a multimeter) across the terminals of the Schottky junction diode and the 9V batteries, as well as other electrical connections in the circuit to ensure that the connections are not faulty, i.e. improperly soldered or connected

- *Electrical insulation of electrical contacts*
    - Ensure that any exposed metallic wire ends are properly insulated for safety purposes
    - Coat sufficient amount of tape – typically used black electrical insulating tape – fully around metallic wire ends
    - Confirm that insulating tape does not disrupt the functioning of other electrical components for testing.

The following **testing instructions** must be followed to test battery functionality according to given specifications.

1. Assemble the circuit based on the aforementioned pre-testing instructions (make sure all components are available)
2. Connect the fuse relay with the power resistors in series, feeding into the power supply and parallel to the dual battery packs (Lithium-ion and LFP)
3. Allow the the thermistor to be in contact with each battery for two seconds to get temperature readings
4. Wire the motor controller directly to the thermocouple digital thermometer for the separate temperature monitoring system accordingly (for alternate temperature sensing, use a thermal IR camera)
5. Setup the multimeter connection point to probe points as close to the motor speed controller as possible (this is due to the voltage drop across the wires at higher amps)
6. Connect the motor controller with the computer by clicking “connect” on the right hand side of the VESC tool, then under the window on the left-hand side, select RT (real-time data) and on top, select the current tab to display the current-time plots for the battery and the motor controller
7. Begin the test experiment by closing the relay in the circuit and manually adjusting *i_motor* (motor current) at the bottom left corner of the VESC tool from 0 A to 15 A, until the battery current *i_batt* reaches 5A in conjunction. Note that VESC software handles the temperature logging of the MOSFETs on the motor controller, and that this can be found on the top under the temperature tab
    a. Record the current readings of each battery pack circuit frequently and continue to monitor the temperature.

3 Operation of the Project

3.1 Operating Mode I: Normal Operation

Discharging

- Carry the battery pack into the eGlider
- Plug the connector to the electric motor
- Check the thermistor to prevent the battery pack from overheating

Charging

- Carry out the battery pack from the eGlider
- Plug the connector from the wall to the battery pack
- Check the thermistors to prevent the battery pack from overheating

3.2 Operating Mode II: Abnormal Operation

Hardware errors

- If the battery cells swell, unplug the charging connector and the discharging connector.
- If the thermistor sensor detects a high temperature inside the battery pack, unplug the connector from the battery to the motor.
- If the battery cells are physically damaged, stop using the battery pack and request repair and replacement services.
- If the battery cell leaks, stop using the battery pack and ask for repair and replacement services.
- If the battery pack is dropped from 1 meter or higher, check whether the battery cells and PCB are in place or not.
- If the battery pack did not charge properly, stop using the battery pack
- If the battery pack did not discharge properly, stop using the battery pack

3.3 Safety Issues

- Users should be careful to not spill liquids into the battery pack. PCB, batteries, and thermistor may be damaged by liquids
- Users should not try to repair the unit by themselves. Contact a local supplier for more details when the packet needs to be replaced
- If the user tries to replace the batteries, replace them with batteries of the same quantity, type, and capacity

Storage

- Battery pack should be stored away from liquids to avoid potential damage. Liquids may cause a short circuit damaging the PCB, batteries, and thermistor.
- Battery pack should be stored away from the combustible materials.
- Battery pack should be stored at temperatures between -40 degrees celsius and 60 degrees celsius .
- Battery pack should be stored away from anything that can catch fire.
- Battery pack should not be stored out of direct sunlight.
- Battery pack should be removed from the device when no longer in use.
- Inspect the status of the battery pack at least weekly.

Charging

- Use a certified input power cable with the correct plugs and sockets to charge the battery pack.
- Do not overcharge or over-discharge the battery.
- Inspect charging signs of the battery pack before charging.
- Check the voltage status of the battery before attaching it to the device.
- Users should only attach the battery pack to the device to satisfy the design requirements. Check the parameter requirements of the device such as power, minimum current flow, maximum current flow, etc.
- Battery pack should be recharged every time it is used

Disposal

- Dispose of a battery pack that no longer holds substantial charge or has physical damage.
- Do not dispose of the battery pack in a regular trash can. Discard the battery pack in a facility capable of cell recycling.
- Do not dispose of the battery pack in an open fire. The battery pack may explode.

4 Technical Background

The battery pack is designed exclusively for eGlider power storage. Any uses outside of this were not taken into consideration during development.

Our approach to solving the problem of eGlider battery storage hinges on the dual chemistry approach of the battery pack. The dual chemistry of our battery pack allows it to use about 80% of its total stored energy during take off as opposed to the industry standard which only uses about 20%. For safety reasons, it is important to use most of a battery pack's stored energy upon launch because in the event of a fire, the temperature and intensity of the fire would be proportional to the remaining energy stored in the battery pack.

We rely on a small pack of LFP 19s1p batteries for early take off in conjunction with a pack of Li-ion batteries which powers the rest of the flight. Both of these batteries were chosen for their specific chemical characteristics. LFP batteries were chosen to power the early takeoff stage because they have a very high continuous discharge rate of ~50A and can supply up to 120A in short bursts. LFP batteries are also very chemically robust, which allows them to handle deep discharges unachievable by Li-ion batteries.

For the late climbing portion of takeoff we rely on a pack of Li-ion 12s8p batteries to provide the bulk of the current, which is expected to be about 200A and decaying down to about 100A over the course of a flight. While LFP will provide fast sourcing, the Li-ion pack’s high energy density will provide the energy needed to reach cruising altitude. Due to the size of the pack, each battery will have ~25A max current on it, which is within their continuous current range.

In addition to the batteries we have also designed supplemental electronic systems to monitor the pack, reduce losses, and notify the pilot. These systems include a temperature sensing system, an ideal diode system, and a pilot interface. We have designed a temperature sensing system for use with a microcontroller carrying ADC units. An issue common with temperature sensing through the use of microcontroller ADCs is that individual thermistors do not scale well. Generally, one ADC pin is needed for each temperature sensor. Microcontrollers typically have around 10 ADC pins, allowing for 10 temperature points on the battery packs. This is insufficient for larger battery packs because a temperature point is needed on every cell to determine cell failures and give advanced warning to the pilot. Our modular approach uses a buffer mosfet and flip-flop ic’s to switch through a string of thermistors using a single ADC. This hardware and code solution solves the ADC problem and has the added benefit of

being modular with the potential to be expanded into industries beyond the scope of our project.

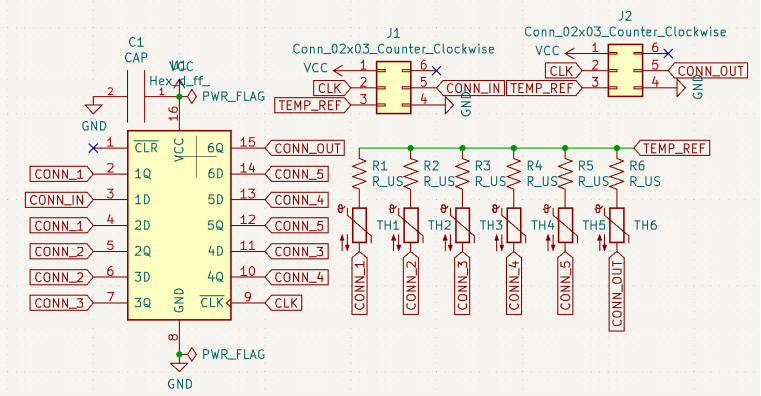


Figure 6: Modular thermistor sampling schematic with connections to micontroller (conn_in, clk, vcc, gnd).

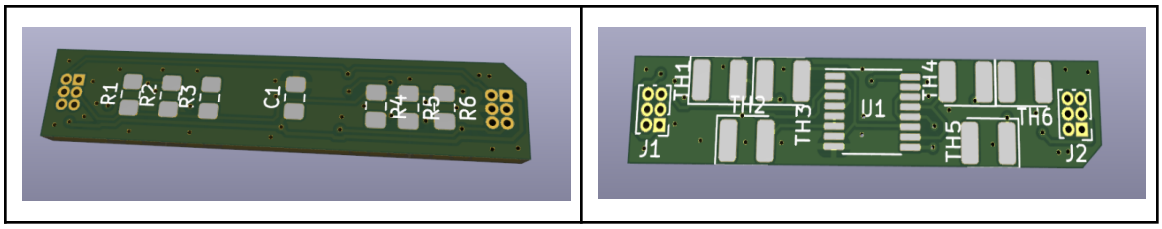


Figure 7: Back (left) and front (right) PCB renders for temperature sensing thermistor modules.

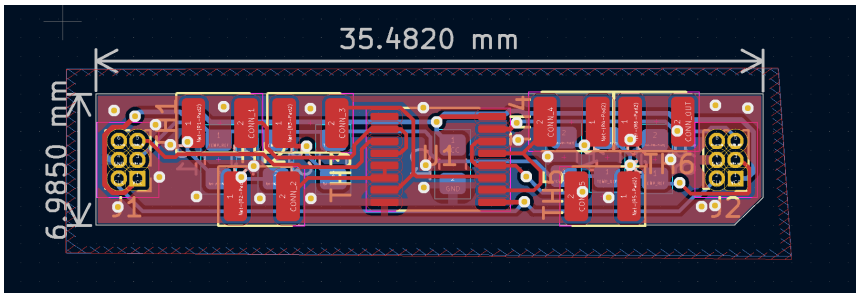


Figure 8: PCB Layout for thermistor temperature sensing modules designed for between cell installation.

Size is the largest issue when it comes to the design of these boards, they are built to fit in between the battery cells in the pack to limit the wire strains and prevent accidental disconnection.

Since we are using a battery configuration of 19s1p for the LFP and 12s8p for the Li-ion battery the nominal voltages are 68.4V and 50.4V respectively, leading to an issue of back feeding current from the LFP batteries to the Li-ion pack. This can be rectified through the use of a diode. One issue with using the diode is power loss, with a high current schottky diode the losses can be expected to surpass 100 Watts. Rather than using a conventional diode we can use what is known as an “ideal diode”. Through the use of an ideal diode controller and three high current mosfets we create a system that works the same way as a diode without the same losses associated. Once the LFP pack reaches the voltage of the Li-ion pack, the ideal diode controller produces a voltage at the gate of the mosfet turning it on and allowing current to flow to the motor controller from the Li-ion batteries.

The PCB for our ideal diode board needs to be designed differently than conventional PCBs. Since the MOSFET array will need to carry up to 300A we cannot use conventional PCB traces to transmit power which generally have a maximum current rating of about 10A. We handle the high current by soldering copper bus bars onto 6 mm traces. High current also leads to the problem of heat dissipation. To manage this problem, there are two copper pours on the top and bottom layers to distribute the heat evenly acting as a heatsink. The PCB also has a conventional heat sink to help with the heat dissipation requirements.

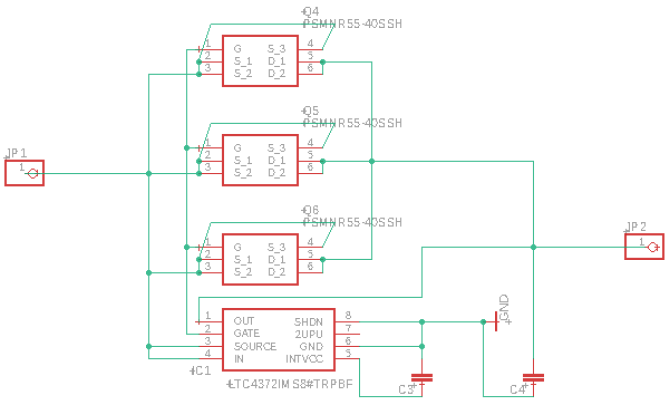


Figure 9: Ideal diode controller PCB schematic.

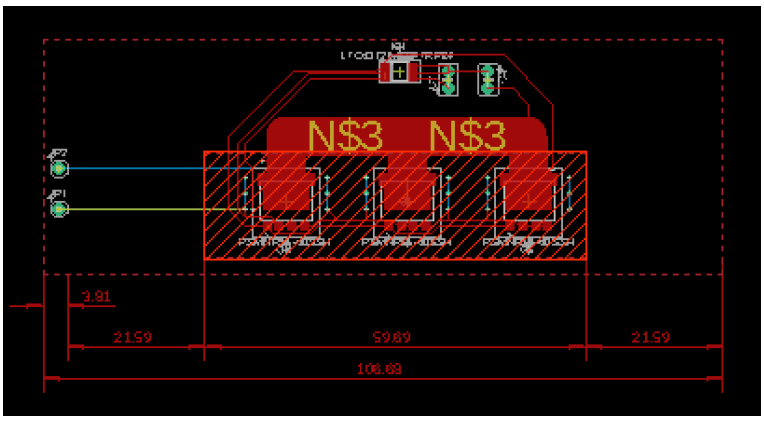


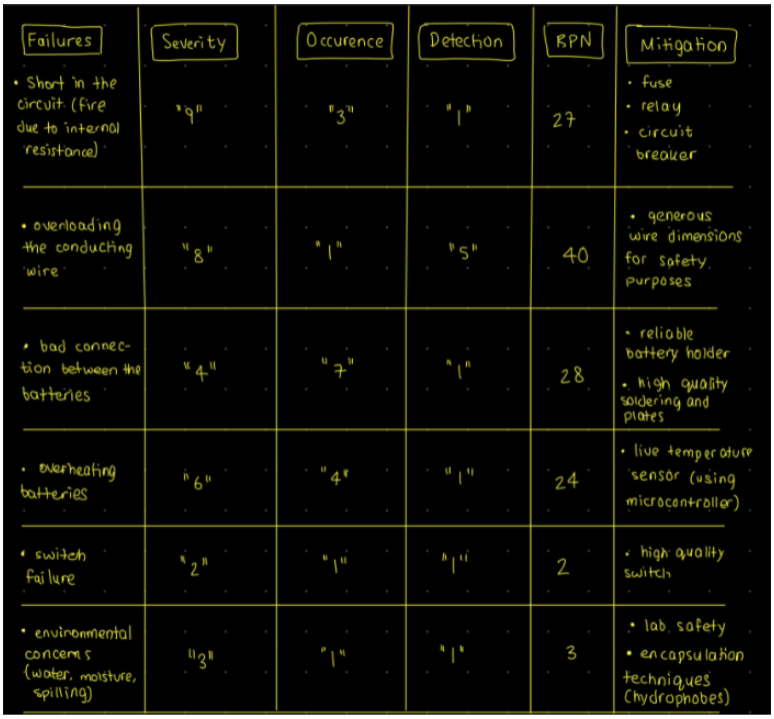
Figure 10: Ideal diode controller PCB board.

To charge the batteries the final pack will require two charging ports, one for each separate chemistry battery pack. These chargers need to be bought from an external vendor and need to be able to handle the voltages and charging schedule needed for each of the two chemistries. This is important because different chemistry batteries require different charging algorithms to ensure the longest battery life and limit degradation.

5 Relevant Engineering Standards

FMEA Analysis

According to the Cambridge Institute for Healthcare Improvement, Failure Modes and Effects Analysis (FMEA) is a proactive method for identifying potential points of failure in a process. The method is used to: (a) evaluate a process to identify where and how it might fail and (b) to assess the relative impact of different failures in order to highlight the elements that pose the greatest amount of risk and consequently that need to be changed. There are several parameters involved: (1) probability that a failure would occur (occurrence), (2) severity of the failure, (3) detection of the failure, (4) risk priority number in accordance with a criticality matrix, and (5) remarks/mitigation/actions. It is necessary to systematically analyze the safety of the first test for the LFP pack before making the full size hybrid battery pack. An integral scale from "1" to "10" has been established for evaluating each of the failure components listed previously according to Figure 11.



Failures	Severity	Occurence	Detection	RPN	Mitigation
• Short in the circuit (fire due to internal resistance)	"9"	"3"	"1"	27	• fuse • relay • circuit breaker
• overloading the conducting wire	"8"	"1"	"5"	40	• generous wire dimensions for safety purposes
• bad connection between the batteries	"4"	"7"	"1"	28	• reliable battery holder • high quality soldering and plates
• overheating batteries	"6"	"4"	"1"	24	• live temperature sensor (using microcontroller)
• switch failure	"2"	"1"	"1"	2	• high quality switch
• environmental concerns (water, moisture, spilling)	"3"	"1"	"1"	3	• lab safety • encapsulation techniques (hydrophobes)

Figure 11: FMEA Analysis table for hybrid battery pack testing.

The risk priority number (RPN) has been calculated using the following equation:

$$RPN = Severity \times Occurrence \times Detection$$

The table illustrates that overloading the conducting wire, bad battery connections, and a possible short in the circuit should take highest priority. The first failure concern can be addressed by providing generous wire dimensions to handle about 550A (calculated from an internal resistance of 6 mΩ and a net voltage of 9.9V, with each battery having a voltage of 3.3V).

Oracle Javadoc Style Programming Documentation

One of the most important factors in any written program is how it is documented and organized. A program that follows a consistent style of both documentation and code formatting is easier to read, understand, debug, and maintain. The Oracle Javadoc is one of the most commonly used documentation styles for large-scale software systems, and as such, this was implemented in the testbench code. In the Lisp scripts, all constants are all fully capitalized (e.g. *TEST-PERIOD* denotes the constant value for the battery testing period in seconds). Whereas, variable identifiers are all set to lowercase letters separated by dashes rather than underscores (e.g. *i-motor* denotes the motor current in Amps). By engineering standards, underscores are used, but Lisp does not permit the use of underscores for variable identifiers as the underscore character is used for pattern matching algorithms, so dashes are the "modified" variable identifier standard for Lisp. Function identifiers follow the same standard as variable identifiers, with the same exception of underscores replaced with dashes in Lisp (e.g. *power-level(flag)* denotes the power level transition function identifier). Every source file has a comment block at the top that contains relevant information. As per Javadoc style, the script contains a brief description of source code's purpose at the top. For the file name, author, affiliation, and version number, the following tags are used respectively: @file [filename], @author [author name], @affiliation [affiliation], and @version [version number]. For generic comment style, three ";" are used to describe the header comment block, global constant group comment blocks, and function definition comment blocks. Then, single-line comments are denoted using single ";". With regards to the function definition comment blocks, the function purpose is first described, then each input parameter the function takes is included on a separate line using the tag @param [parameter name] [parameter description]. The output parameters (or the values returned by the function) are denoted, also on separate lines, using the tag @return [return name] [return description].

6 Cost Breakdown

Project Costs for Production of Beta Version[1]				
Item	Quantity	Description	Unit Cost ($)	Extended Cost ($)
1	19	$LiFePO_4$	14	266
2	96	Li-ion	8.53	818.56
5	1	Ideal Diode Controller	7.9	7.90
6	10 ft	4/0 AWG Wires (5 ft)	50	100
7	21	PCB Battery Temperature Sensing	1.65	34.65
8	1	1 μF Capacitor for ideal Diode	0.28	0.28
9	1	MOSFET for Ideal Diode	6.44	6.44
10	1	0.1 μF Capacitor for ideal Diode	1.07	1.07
11	1	Uxcell NTC Thermistors Resistor MF52-103/3435 10K Ohm Temperature Sensor Pack of 60	9.99	9.99
12	1	Adafruit Huzzah ESP32	24.61	24.61
Total Cost ($)				1,237.10

The beta product estimated cost breakdown of the hybrid battery pack will consist of all hardware components required to realize the full version of our battery design.

The main expense of the project will be coming from the two different battery cells, LFP 19s1p setup ($266) and Li-ion 12s8p ($818.56). The rest of the product design component will have a diode controller ($7.90), thermistors ($9.99), an Adafruit Huzzah ESP32 ($24.61) and two different PCBs – one for the ideal diode ($7.50) and 20 for the temperature sensing of the battery pack ($34.65). The battery pack will also be wired with a stronger kind capable of handling 300A rated between 2/0 - 4/0 AWG (~$100).

As for software we use to make the heatmap display we are using an open source software application called Qt creator which is free with no additional cost.

[1] The beta version is the third prototype (the product that comes after the second prototype testing).

7 Appendices

7.1 Appendix A - Specifications

Requirement value, range, tolerance, units	Requirement value, range, tolerance, units
Temperature and Environment	
Environmentals	- function between 0°C and 40°C - withstand temperatures between -20°C and 50°C
Normal discharge temperatures	- upper limit of 70°C - preference of 50°C
Target final SoC	- 5% for the LFP - 20% for the Li-ion
Energy	
Self-launch time	- ~ 4 minutes
Target power profile	- 0s to 60s at >16 kW - 60s to 180s at ~12 kW - 180s to 300s at ~10 kW - 300s to 600s at ~6 kW
Target final SoC	- 5% for the LFP - 20% for the Li-ion
Power connector	- must handle 300A for 60s
Target voltage	- 60V peak under load
Battery Management System	- two different BMS for each pack
Physical Quantity	
Target weight	- Goal: 10kg - Threshold: 12kg

Charging and Discharging	
Initial discharge rate	- 300A
Final discharge rate	- 100A
Target cycle life	- threshold: 500 launches - goal: 1000 cycles
Precharging	- handled by exterior components
Charge connectors	- each cell must be individually accessible for balancing purposes

7.2 Appendix B – Team Information

Team #23: HYP Batteries consists of Spencer P. Piligian, Daniel Kao, Yusheng Chen, Steven Wang, and Ismaeel AlAlawi. We are a team of five electrical engineers working on HYP batteries. This project aims to help us further develop our technical skills and knowledge in both the hardware & circuit design space and embedded software prototyping.

Post-Graduation Plans:

Spencer P. Piligian
B.S Electrical Engineering
Career: General Motors TRACK Engineer

Daniel Kao
B.S Electrical Engineering
Career: Undecided

Yusheng Chen
B.S Electrical Engineering
Career: Master's at Columbia University

Steven Wang
B.S Electrical Engineering
Career: General Dynamics, Systems Engineer II

Ismaeel AlAlawi
B.S Electrical Engineering
Career: Undecided